Ministère de l'Enseignement Supérieur et
de la Recherche Scientifique
Institut International de Technologie à Sfax

وزارة التعليم العالي والبحث العلمي
المدرسة العليا الدولية الخاصة للتكنولوجيا بصفا قس

PROJET DE STAGE D'ÉTÉ

Effectué à

CLINI SYS

Préparé par

Yassmine Kharrat

Licence en génie logiciel et système d'information

Intitulé :

RÉALISATION D'UNE APPLICATION SPRING BOOT POUR LA GESTION DES SERVEURS

Année universitaire 2021-2022

Remerciement

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé leur de la rédaction de ce rapport

Tout d'abord, j'adresse mes remercîments à Mr Marwen Hadrich qui m'a

Beaucoup aider dans ma recherche de stage et m'a permis de postuler dans cette entreprise

Je remercie également toute l'équipe de CliniSys pour leur accueil et leur esprit d'equipe

Enfin je tiens à remercier ma famille pour leurs encouragement et soutien

Table des matières

Introduction

Dans le cadre de ma licence en génie logiciel et système d'information à l'institut international technologie de Sfax, j'ai souhaité réaliser dans une entreprise répondant à mes besoins de futur en spring boot.

Clinisys, l'entreprise que je l'ai fait mon stage, m'a aidé beaucoup à connaitre le spring boot
Ce stage a duré 4 semaines

L'objectif du stage :
-le développement en java

Présentation de la société

1.1. Présentation générale :

Clinisys est une société de service et d'ingénierie en informatique SSII.

Elle a été créée, depuis 1991, sous l'appellation COMPUTER SYSTEM certifiée ISO 9001- 2015 Depuis 2017.

Figure 1.*Logo de Clinisys*

Son effectif total est :

- 140 personnes dont 100 ingénieurs, ingénieurs consultants, développeurs et techniciens supérieurs.
- 23 techniciens maintenance

La plupart des clients conventionnés avec clinisys sont des établissements sanitaires tel que hôpitaux, polycliniques, centres de radiologie, laboratoires d'analyses, pharmacies dans toute la Tunisie et aussi à l'étranger : Maroc, égypt., Libye et aussi autre secteurs industriels.

2.2. Activité :

- Pour les hôpitaux, clinique, centre :

-les programmes des cliniques càd la création des bases de données et des logiciels
- développement des solutions informatiques
-sécurité des différentes infrastructures du projet
- des formations agréer par le ministère de l'éducation de la formation
- la comptabilité et le finance

- Centre de radiologie RIS/PACS :

-tableau de bord radiologie

- Gestion Laboratoire LIS :

-intégration directe avec les cliniques

3.3. Clinisys ERP de santé :

CLINISYS ERP est une solution de gestion hospitalière totalement intégrée spécialement conçue afin de couvrir l'intégralité des fonctions des établissements de santé.

Figure 2.*Logo de Clinisys ERP de santé*

Présentation du travail effectué

DESCRIPTION DU SUJET

Le travail effectué durant ce stage consiste à la réalisation des tâches suivantes :

-la création d'une base de données

-le codage en java avec Spring Boot

- Les logiciels utilisés lors de la réalisation des tâches sont :

. Microsoft SQL Server Management Studio :

Figure 3.*Logo du SSMS*

Est un logiciel pour configurer, gérer et administrer les moteurs de base de données SQL Server

. Apache NetBeans IDE 12.1 :

Figure 4.*Logo de NetBeans*

NetBeans est un environnement de développement intégré (IDE) open-source pour le développement avec Java, PHP, C++ et d'autres langages de programmation.

- Le Langage utilisé :JAVA

Figure 5.*Logo de java*

Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton

- Le Framework: Spring boot

Figure 6.*Logo de Spring Boot*

Est un Framework Java open source utilisé pour créer un micro-Service.

Un Micro-Service est une architecture qui permet aux développeurs de développer et de déployer des services de manière indépendante.

Chaque service en cours d'exécution a son propre processus, ce qui permet d'obtenir le modèle léger pour prendre en charge les applications métier.

Ce Framework a été développé par Pivotal Team et utilisé pour construire des applications autonomes et prêtes à la production.

DESCRIPTION DES TACHES

1.1. Création des tables SQL :

Dans ce stage, j'ai créé des tables SQL dans une base nommée GestionServers

Cette BD contient les tables suivantes :

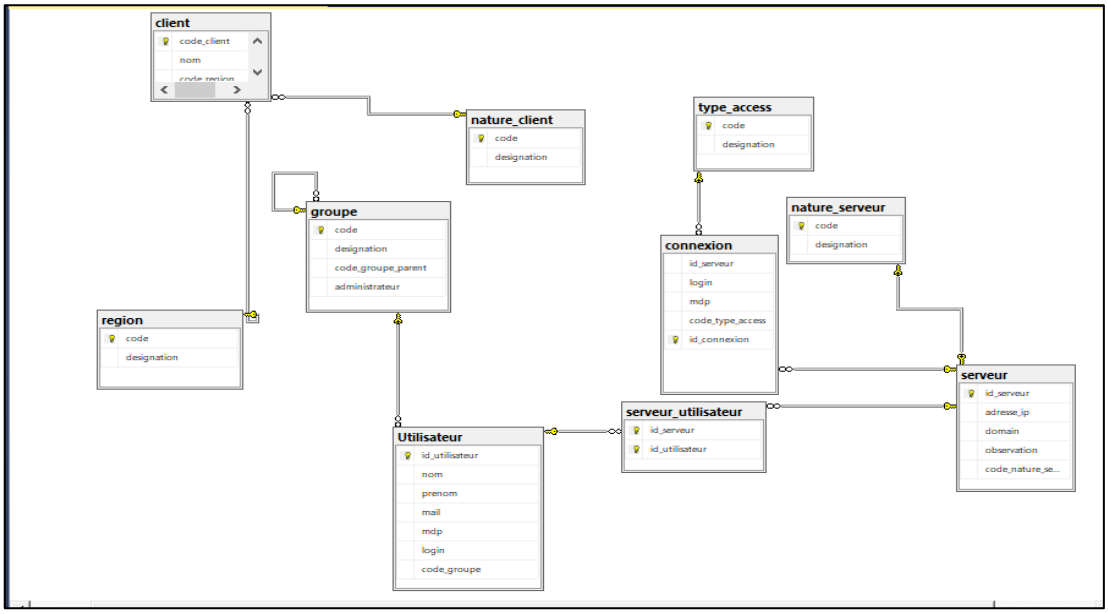

Figure 7.*GestionServers*

2.2. Implémentation de ces tables à partir du spring boot :

- Pour établir une connexion à cette base de données sur Microsoft SQL Server :

✓ déclarez une dépendance pour le pilote JDBC SQL Server qui permet à l'application Java de se connecter à Microsoft SQL Server.

```
<dependency>
    <groupId>com.microsoft.sqlserver</groupId>
    <artifactId>mssql-jdbc</artifactId>
    <scope>runtime</scope>
</dependency>
```

Figure 8.*Dépendance pour SQL Server*

✓ Spécifier les propriétés de la source de données

```yaml
spring:
    devtools:
        restart:
            enabled: true
        livereload:
            enabled: true
    datasource:
#         url: jdbc:sqlserver://192.168.0.42\sql2008r2;database=ComptaMAN
#         username: SA
#         password: 123
        url: jdbc:sqlserver://192.168.1.246\sql2008r2;database=ComptaNAB
#         url: jdbc:sqlserver://DESKTOP-O0BIJ58;database=ComptaMAN
        username: SA
        password: m0dp@a$$CSYS
```

Figure 9.*Application-dev.yml*

- ✓ Connectez-vous à SQL server avec Spring Data Jpa

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Figure 10.*Dépendance pour Spring Jpa*

2.2.1. Package domain :

Pour utiliser ses tables à partir d'une application spring boot,

Ici Spring boot fait l'implémentation automatique de ces classes

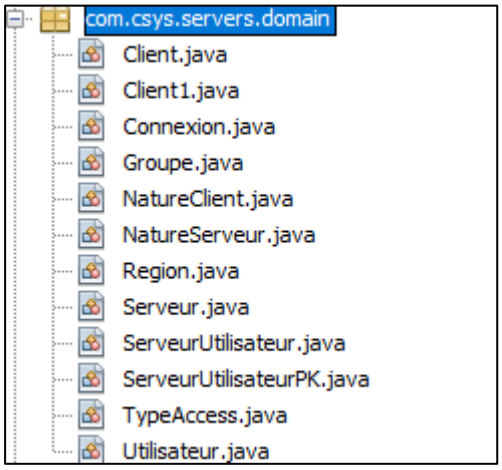

Figure 11. *Package Domaine*

- Pour la classe client :

```java
@Entity
@Table(name = "client")
@NamedQueries({
    @NamedQuery(name = "Client.findAll", query = "SELECT c FROM Client c")})
public class Client implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "code_client", nullable = false)
    private Integer codeClient;
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 50)
    @Column(name = "nom", nullable = false, length = 50)
    private String nom;
    @JoinColumn(name = "code_nature", referencedColumnName = "code", nullable = false)
    @ManyToOne(optional = false)
    private NatureClient natureClient;
    @JoinColumn(name = "code_region", referencedColumnName = "code", nullable = false)
    @ManyToOne(optional = false)
    private Region region;
    public Client() { }
    public Client(Integer codeClient) {
        this.codeClient = codeClient;}
    public Client(Integer codeClient, String nom) {
        this.codeClient = codeClient;
        this.nom = nom;}
```

Figure 12.*Class Client*

- Lorsqu'il y a plus qu'une clé primaire on doit utiliser @JoinColumn et le pk de la classe Comme la classe ServeurUtilisateur :

```java
@Entity
@Table(name = "ServeurUtilisateur")
@NamedQueries({
    @NamedQuery(name = "ServeurUtilisateur.findAll", query = "SELECT n FROM ServeurUtilisateur n")})
public class ServeurUtilisateur implements Serializable {
    private static final long serialVersionUID = 1L;
    @EmbeddedId
    private ServeurUtilisateurPK serveurUtilisateurPK;
    @JoinColumn(name = "idServeur", referencedColumnName = "idServeur", nullable = false)
    @ManyToOne(optional = false)
    private Serveur serveur;
    @JoinColumn(name = "idUtilisateur", referencedColumnName = "idUtilisateur", nullable = false)
    @ManyToOne(optional = false)
    private Utilisateur utilisateur;
    public ServeurUtilisateurPK getServeurUtilisateurPK() {
        return serveurUtilisateurPK;
    }
    public void setServeurUtilisateurPK(ServeurUtilisateurPK serveurUtilisateurPK) {
        this.serveurUtilisateurPK = serveurUtilisateurPK;
    }
    public ServeurUtilisateur() { }
    public Serveur getServeur() {
        return serveur;   }
    public void setServeur(Serveur serveur) {
        this.serveur = serveur;}
    public Utilisateur getUtilisateur() {
        return utilisateur;}
    public void setUtilisateur(Utilisateur utilisateur) {
        this.utilisateur = utilisateur;
    }
}
```

Figure 13.Class ServeurUtilisateur

La figure suivante montre l'implémentation de la classe ServeurUtilisateurPK qui utilise les deux attributs id_serveur et id_utilisateur :

```java
@Embeddable
public class ServeurUtilisateurPK implements Serializable {
    @Basic(optional = false)
    @NotNull
    @Column(name = "id_serveur", nullable = false)
    private Integer idServeur;
    @Basic(optional = false)
    @NotNull
    @Column(name = "id_utilisateur", nullable = false)
    private Integer idUtilisateur;
    public Integer getIdServeur() {
        return idServeur;}
    public void setIdServeur(Integer idServeur) {
        this.idServeur = idServeur;}
    public Integer getIdUtilisateur() {
        return idUtilisateur;}
    public void setIdUtilisateur(Integer idUtilisateur) {
        this.idUtilisateur = idUtilisateur;}
    @Override
    public int hashCode() {
        int hash = 7;
        hash = 29 * hash + Objects.hashCode(this.idServeur);
        hash = 29 * hash + Objects.hashCode(this.idUtilisateur);
        return hash;}
```

Figure 14.*Class ServeurUtilisateurPK*

2.2.2. Package dto :

Sans DTO, nous devrions exposer l'ensemble des entités à une interface distante. Cela provoque un couplage fort entre une API et un modèle de persistance.

En utilisant un DTO pour transférer uniquement les informations requises, nous desserrons le couplage entre l'API et notre modèle, ce qui nous permet de maintenir et de mettre à l'échelle plus facilement le service

La figure suivante montre tous les classes DTO :

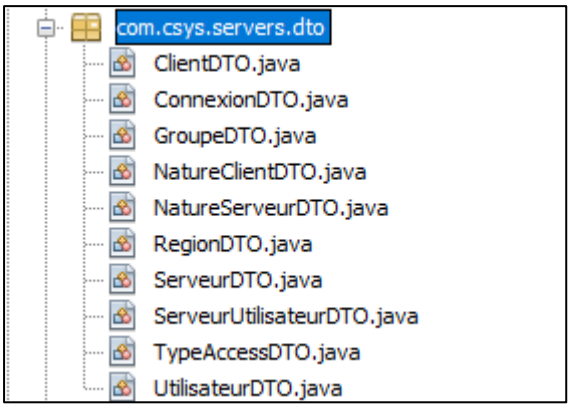

Figure 15.*Package dto*

Pour la classe ClientDTO :

```java
public class ClientDTO {

    private Integer codeClient;

    @NotNull
    @Size(min = 1, max = 50)
    private String nom;

    private RegionDTO region;

    private NatureClientDTO natureClient;

    private Integer codeNature;
    private Integer codeRegion;


    public ClientDTO() {
    }

    public Integer getCodeClient() {
        return codeClient;
    }

    public void setCodeClient(Integer codeClient) {
        this.codeClient = codeClient;
    }

    public String getNom() {
        return nom;
    }
```

Figure 16.*Class ClientDTO*

2.2.3. Package factory :

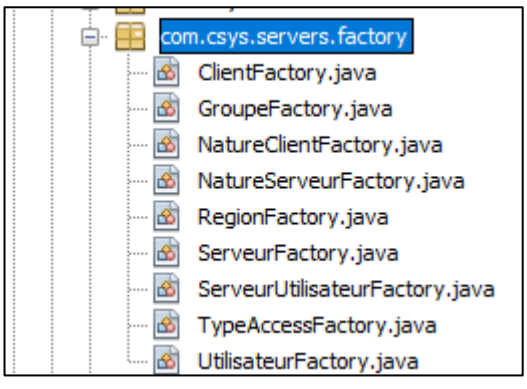

Figure 17.*Package factory*

Pour la classe clientFactory :

```
public class ClientFactory {

    public static ClientDTO clientToClientDTO(Client client) {

        ClientDTO clientDTO = new ClientDTO();
        clientDTO.setCodeClient(client.getCodeClient());
        clientDTO.setNom(client.getNom());
        clientDTO.setRegion(RegionFactory.regionToRegionDTO(client.getRegion()));
        clientDTO.setNatureClient(NatureClientFactory.natureClientToNatureClientDTO(client.getNatureClient()));
        return clientDTO;
    }

    public static Client clientDTOToClient(ClientDTO clientDTO, NatureClient natureClient, Region region) {

        Client client = new Client();
        client.setCodeClient(clientDTO.getCodeClient());
        client.setNom(clientDTO.getNom());
        client.setRegion(region);
        client.setNatureClient(natureClient);
        return client;
    }

    public static List<ClientDTO> ClientToClientDTOs(List<Client> clients) {
        List<ClientDTO> clientsDTO = new ArrayList<>();
        clients.forEach(x -> {
            clientsDTO.add(clientToClientDTO(x));
        });
        return clientsDTO;
    }

}
```

Figure 18.*Class ClientFactory*

2.2.4. Package repository :

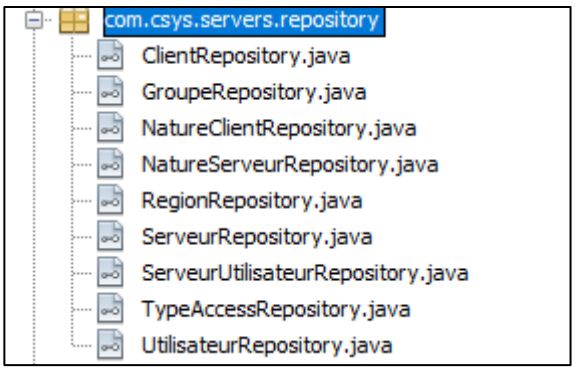

Figure 19.*Package repository*

Pour la classe clientRepository :

```
public interface ClientRepository extends JpaRepository<Client, Integer> {

    Optional<Client> findOneByNom(String nom);

    public Client findOne(Integer idClient);

}
```

Figure 20.Class ClientRepository

2.2.5. Package service :

Le package service contient toutes les implémentations des services :

Figure 21.*Package service*

Par exemple pour la classe clientService :

- Pour rechercher la liste de tous les client on va écrire une fonction avec la méthode getAllClient() comme suit :

```
@Service
@Transactional
public class ClientService {

    private final Logger log = LoggerFactory.getLogger(ClientService.class);
    private final ClientRepository clientRepository;
    private final NatureClientRepository natureClientRepository;
    private final RegionRepository regionRepository;

    public ClientService(ClientRepository clientRepository, NatureClientRepository natureClientRepository, RegionRepository regionRepository) {
        this.clientRepository = clientRepository;
        this.natureClientRepository = natureClientRepository;
        this.regionRepository = regionRepository;
    }

    @org.springframework.transaction.annotation.Transactional(readOnly = true)
    public List<ClientDTO> getAllClient() {
        List<Client> client = clientRepository.findAll();
        return ClientFactory.ClientToClientDTOs(client);
    }
```

Figure 22.*Constructeur et Fonction getAllClient*

Cette fonction utilise la classe clientDTO pour communiquer avec les attributs

et recherche la liste à partir de l'interface clientRepository

- Pour chercher un seul client à partir de son id , on utilise la fonction par défaut findOne comme suit :

```
@org.springframework.transaction.annotation.Transactional(readOnly = true)
public ClientDTO findOne(Integer idClient) {
    Client client = clientRepository.findOne(idClient);
    return ClientFactory.clientToClientDTO(client);

}
```

Figure 23.*Fonction findOne*

- On fait appel à une fonction par défaut save comme suit :

```
public ClientDTO save(ClientDTO clientDTO) {
    log.debug("Request to save Client: {}", clientDTO);
    Client client = clientRepository.findOne(clientDTO.getCodeClient());
    Preconditions.checkBusinessLogique(client == null, "Le client est déja existant");
    NatureClient natureClient = null;
    if (clientDTO.getCodeNature() != null) {
        natureClient = natureClientRepository.findOne(clientDTO.getCodeNature());
        Preconditions.checkBusinessLogique(natureClient != null, "la nature du client n'existe pas");
    }
    Region region = null;
    if (clientDTO.getCodeRegion() != null) {
        region = regionRepository.findOne(clientDTO.getCodeRegion());
        Preconditions.checkBusinessLogique(region != null, "la région n'existe pas");
    }
    Client clt = ClientFactory.clientDTOToClient(clientDTO, natureClient, region);
    clt = clientRepository.save(clt);
    ClientDTO resultDTO = ClientFactory.clientToClientDTO(clt);
    return resultDTO;
}
```

Figure 24.*Fonction save*

- Pour vérifier l'existence du client, on fait appel à la fonction update comme suit :

```java
public ClientDTO update(ClientDTO clientDTO) {
    log.debug("Request to save Client: {}", clientDTO);
    Client client = clientRepository.findOne(clientDTO.getCodeClient());
    Preconditions.checkBusinessLogique(client != null, "Le client n'existe pas");
    NatureClient natureClient = null;
    if (clientDTO.getCodeNature() != null) {
        natureClient = natureClientRepository.findOne(clientDTO.getCodeNature());
        Preconditions.checkBusinessLogique(natureClient != null, "la nature du client n'existe pas");
    }
    Region region = null;
    if (clientDTO.getCodeRegion() != null) {
        region = regionRepository.findOne(clientDTO.getCodeRegion());
        Preconditions.checkBusinessLogique(region != null, "la région n'existe pas");
    }
    client = ClientFactory.clientDTOToClient(clientDTO, natureClient, region);
    client = clientRepository.save(client);
    ClientDTO resultDTO = ClientFactory.clientToClientDTO(client);
    return resultDTO;
}
```

Figure 25.*Fonction update*

- Pour supprimer un client, la fonction delete est comme suit :

```java
public Boolean delete(Integer codeClient) {
    log.debug("Request to delete client: ", codeClient);
    Client client = clientRepository.findOne(codeClient);
    Preconditions.checkBusinessLogique(client != null, "Le client n'existe pas");
    clientRepository.delete(codeClient);
    return true;
}
}
```

Figure 26*.Fonction delete*

2.2.6. Package web rest :

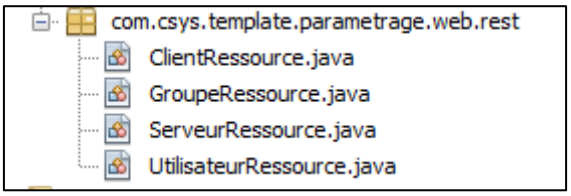

Figure 27.*Package web rest*

```java
@RestController
@RequestMapping("/api/")
public class ClientRessource {

    @Autowired
    private ClientService clientService;

    @GetMapping("/Client")
    public List<ClientDTO> getAllClient() {
        return clientService.getAllClient();
    }

    @PostMapping("/Client")
    public ClientDTO createClient(@RequestBody ClientDTO clientDTO) {
        return clientService.save(clientDTO);
    }

    @PutMapping("/Client")
    public ClientDTO updateClient(@RequestBody ClientDTO clientDTO) {
        return clientService.save(clientDTO);
    }

    @DeleteMapping("/Client")
    public Boolean deleteClient(@RequestParam(name = "codeClient", required = true) Integer codCli) {
        return clientService.delete(codCli);
    }
}
```

Figure 28.*Class ClientRessource*

Conclusion

L’objectif principal de ce stage était la découverte du monde de l’entreprise et dans cette optique ce stage a totalement répondu à mes attentes

Il convient de souligner un autre point important qui m’a permis une adaptation rapide de ce nouveau contexte : c’est la confiance qu’on m’a accordé à CliniSys dès mon arrivée. En fait, tous au long de ce stage j’ai eu le plaisir de travailler en autonomie ce qui m’a obligé à prendre quelques initiatives et à rechercher. De plus, j’ai appris à gérer mon temps afin de faire le maximum de travail prévu pour fournir l’aide nécessaire à l’entreprise.

Finalement, je n’ai qu’a exprimer ma satisfaction envers mes relations avec tout le personnel du site qui m’ont aidé durant toute la période de stage.